C/C++ Pointers vs References

Consider the following code:

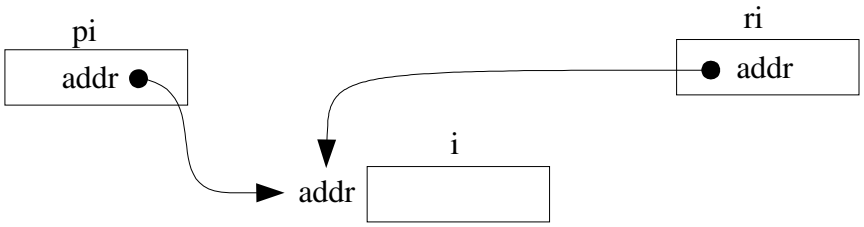
Pointers

```
int i;
int *pi = &i;
```

References

```
int i;
int &ri = i;
```

In both cases the situation is as follows:



Both `pi` and `ri` contain addresses that point to the location of `i`, but the difference lies in the appearance between references and pointers when they are used in expressions. In order to assign a value of 4 to i in both cases, we write:

```
*pi = 4;
```

```
ri = 4;
```

Note the, when using pointers, the address must be dereferenced using the *, whereas, when using references, the address is dereferenced without using any operators at all!

The main effect of this is that the address can directly be manipulated if it is a pointer. We can do things such as:

```
pi++;
```

to increment to the next address. This is not possible using references. Therefore, to summarize, a pointer can point to many different objects during its lifetime, a reference can refer to only one object during its lifetime.

When to Use Pointers vs References

References are the preferred way of indirectly accessing a variable. They are also a little safer than pointers and, in some cases, are the only way to achieve a particular result such as overloading certain operators. Consider the following:

```
enum day
{
Sunday, Monday, ...
};
```

If we define a variable;

```
day x;
```

and we wanted to write a method to overload the ++ operator such that the statement

```
++x;
```

increments `x` to the next day, we could write the following:

```
day &operator++(day &d)
{
d = (day)(d + 1);
return d;
}
```

Using pointers, we may think that the following declaration would work:

```
day *operator++(day *d);
```

However, this statement will not even compile because every overloaded operator function must either be a member of a class, or have a parameter of type T, T &, or T const &, where T is a class or enumeration type. So in this particular case, using a reference is the only way to do it.